Beehive Page Flow Tutorial

Table of contents

1. Introduction

1.1. Tutorial Goals

In this tutorial, you will learn:

- how to create a basic Page Flow web application.
- how to coordinate user navigation with Forward methods.
- how to handle data submission and processing with data binding and Form Beans.
- how to create a user interface with the <netui> JSP tag library.
- how Page Flows help to separate data processing and data presentation.
- how to use declarative validation with data submission.
- how to collect data from a nested Page Flow and 'return' it to the nesting Page Flow.
- how to make an action available to multiple Page Flows.

2. Step 1: Begin the Page Flow Tutorial

2.1. To Set up the Development Environment

Complete all of the necessary and optional steps in the following topic: Beehive Installation and Setup (../setup.html)

Open a command shell and confirm that you have set the following variables:

- `ANT_HOME`
- `JAVA_HOME`
- `CATALINA_HOME`

Also ensure that the following elements are on your `PATH`:

- `ANT_HOME/bin`
- `JAVA_HOME/bin`

2.2. Make a Project Folder

On your C: drive, create a directory named `beehive_projects`.

Copy the folder `<BeehiveRoot>/samples/netui-blank` into `C:/beehive_projects`. (`<BeehiveRoot` is the top level folder of your Beehive installation; a typical value might be `C:/apache/apache-beehive-1.0`.)

Rename the folder `C:/beehive_projects/`**`netui-blank`** to the name `C:/beehive_projects/`**`pageflow_tutorial`**

Before proceeding, confirm that the following directory structure exists:

```
C:
  beehive_projects
    pageflow_tutorial
      resources
      WEB-INF
      Controller.java
      index.jsp
```

2.3. Edit the build.properties File

In this section you will edit the `build.properties` file -- the file that sets the build-related properties for your web application.

Open the file `C:/beehive_projects/pageflow_tutorial/WEB-INF/src/build.properties` in a text editor.

Edit the `beehive.home` property so it points to the top-level folder of your beehive installation.

Edit the `context.path` so it has the value `pageflow_tutorial`.

> **Note:**
> The `context.path` property determines both (1) the name of the application WAR file and (2) the application URL.
> If `context.path=myWebApp`, then the following WAR file will be produced:
> **`myWebApp`**`.war`
> and the following URL will invoke the web application:
> `http://someserver/`**`myWebApp`**

For example, if your beehive installation resides at `C:/apache/apache-beehive-1.0`, then your `build.properties` file would appear as follows.

```
beehive.home=C:/apache/apache-beehive-1.0

servlet-api.jar=${os.CATALINA_HOME}/common/lib/servlet-api.jar
jsp-api.jar=${os.CATALINA_HOME}/common/lib/jsp-api.jar

context.path=pageflow_tutorial
```

> **Note:**
> Windows users must use forwardslashes (/) not backslashes (\) in the `build.properties` file.

2.4. To Start the Tomcat Server

At the command prompt, enter:

```
%CATALINA_HOME%\bin\startup.bat
```

3. Step 2: Create a New Page Flow Application

3.1. Introduction

In this step you will create a Controller file and a JSP page. These are the basic files in a Beehive Page Flow web application. Each Page Flow contains one Controller file and any number of JSP pages. A Controller file is a Java class that controls how your web application functions and what it does. The methods in the Controller file determine all of the major features of a web application: how users navigate from page to page, how user requests are handled, and how the web application accesses back-end resources. The JSP pages determine what a visitor to the web sees in the browser.

In terms of the Model-View-Controller paradigm for web applications, the Controller.java file is the Controller (naturally) and the JSP pages are the View. The web application's Model in this tutorial is very simple: it consists of three fields that represent the user's name, age and selected sport activity.

Controller files contain Action methods. An Action method may do something simple, such as forward a user from one JSP page to another; or it may do a complex set of tasks, such as receive user input from a JSP page, calculate and/or retrieve other data based on the user input, and forward the user to a JSP page where the results are displayed.

The Controller file in this step contains one simple Action method. This simple navigational Action method forwards users to the index.jsp page. In the next step, you will create a more complex Action method.

3.2. To Examine the Controller.java and index.jsp Files

There are no edits to make in this step. The point of this step is to learn about the code you are about to run.

Open the file `C:/beehive_projects/pageflow_tutorial/Controller.java`.

The Controller file is an ordinary Java class with methods and annotations.

A Page Flow controller class must extend `org.apache.beehive.netui.pageflow.PageFlowController` and be decorated by the annotation `@Jpf.Controller`.

The `onCreate()` method is executed whenever the Controller class is first instantiated. The `onDestroy()` method is executed when the Controller class is destroyed.

After the `onCreate()` method is run, the Page Flow runtime searches for (and runs) a method or action named `begin`. In this Controller file, there is a simple action named `begin`:

```
@Jpf.SimpleAction(name="begin", path="index.jsp")
```

The begin action *could* have been expressed using method syntax:

```
@Jpf.Action(
    forwards = {
        @Jpf.Forward(name="success", path="index.jsp")
    }
)
public Forward begin()
{
    return new Forward("success");
}
```

We have used the Simple Action syntax for the sake of syntactical simplicity. The Simple Action will forward the user to the JSP, index.jsp.

The Controller class is instantiated when a user calls it via the URL:

```
http://localhost:8080/pageflow_tutorial/begin.do
```

The URL above means this: "Run the `begin` action of the `Controller.java` class in the directory `pageflow_tutorial`."

`Controller.java`

```
import javax.servlet.http.HttpSession;

...

import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    },
    sharedFlowRefs={
        @Jpf.SharedFlowRef(name="shared", type=shared.SharedFlow.class)
    }
)
public class Controller
    extends PageFlowController
{
    @Jpf.SharedFlowField(name="shared")
    private shared.SharedFlow sharedFlow;
```

```
    /**
     * Callback that is invoked when this controller instance is created.
     */
    protected void onCreate()
    {
    }

    /**
     * Callback that is invoked when this controller instance is destroyed.
     */
    protected void onDestroy(HttpSession session)
    {
    }
}
```

Open the file `C:/beehive_projects/pageflow_tutorial/index.jsp`.

`index.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0"
prefix="netui-data"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-template-1.0"
prefix="netui-template"%>
<netui:html>
  <head>
    <title>Web Application Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      New Web Application Page
    </p>
  </netui:body>
</netui:html>
```

3.3. To Compile and Deploy the Page Flow

You are now ready to compile the Page Flow and deploy it to Tomcat.

At the command prompt, enter:

```
ant
  -f C:\beehive_projects\pageflow_tutorial\WEB-INF\src\build.xml
  clean
  build
  war

Copy and Paste version:
ant -f C:\beehive_projects\pageflow_tutorial\WEB-INF\src\build.xml clean
build war
```

To deploy the application to Tomcat, copy the WAR file into Tomcat's `webapps` directory.

```
copy C:\beehive_projects\pageflow_tutorial.war %CATALINA_HOME%\webapps /Y
```

3.4. To Test the Page Flow Web Application

Visit the following address:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the `index.jsp` page.

4. Step 3: Navigation

4.1. To Create a Destination JSP Page

In the directory `C:/beehive_projects/pageflow_tutorial`, create a file named `page2.jsp`.

Edit page2.jsp so it appears as follows.

`page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
  </netui:body>
</netui:html>
```

Save `page2.jsp`.

4.2. To Create a Link to the Destination Page

In this step you will create a link from the JSP, `index.jsp` to a new Simple Action that you will add to the Controller file.

Open the file `C:/beehive_projects/pageflow_tutorial/index.jsp`.

Edit `index.jsp` so it appears as follows. The code to add appears in bold type.

index.jsp

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0"
prefix="netui-data"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-template-1.0"
prefix="netui-template"%>
<netui:html>
  <head>
    <title>Web Application Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      New Web Application Page
    </p>
    <p>
      <netui:anchor action="toPage2">Link to page2.jsp</netui:anchor>
    </p>
  </netui:body>
</netui:html>
```

Save `index.jsp`.

4.3. To Add a Simple Action to Handle the Link

Open the file `C:/beehive_projects/pageflow_tutorial/Controller.java`.

Edit `Controller.java` so it appears as follows. Don't forget the comma after the first `Jpf.SimpleAction(...)` element!

Controller.java

```
import javax.servlet.http.HttpSession;

...

import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    },
    sharedFlowRefs={
        @Jpf.SharedFlowRef(name="shared", type=shared.SharedFlow.class)
    }
```

```
)
public class Controller
    extends PageFlowController
{
    ...
}
```

Save `Controller.java`.

4.4. To Recompile and Redeploy the Page Flow

Compile and deploy the Page Flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

4.5. To Test the Page Flow Web Application

Visit the following link:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the index.jsp page.

Click the link.

You will be directed to page2.jsp.

5. Step 4: Submitting Data

5.1. To Create a Submission Form

This step illustrates the use of custom tags to render an HTML form tag and link it to an Action. In a later step, the new Action will be added to the Controller file to handle the data submission.

Edit the file `C:/beehive_projects/pageflow_tutorial/page2.jsp` so it appears as follows.

`page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
```

```
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
    <p>
      <netui:form action="processData">
        Name: <netui:textBox dataSource="actionForm.name"/>
        <br/>
        Age: <netui:textBox dataSource="actionForm.age"/>
        <br/>
        <netui:button type="submit" value="Submit"/>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save `page2.jsp`.

5.2. To Create a Server Side Representation of the Submission Form (a.k.a. Create a Form Bean)

In this step you will create a Java class that represents the submission form created in the previous task. When the form data is submitted, the Java class will be instantiated, and the form data will be loaded into the members of the Java class.

In the directory `C:/beehive_projects/pageflow_tutorial/WEB-INF/src` create a directory named **`forms`**.

In the directory `C:/beehive_projects/pageflow_tutorial/WEB-INF/src/forms` create a JAVA file named **`ProfileForm.java`**.

Edit `C:/beehive_projects/pageflow_tutorial/WEB-INF/src/forms/ProfileForm.java` so it appears as follows.

`ProfileForm.java`

```
package forms;

public class ProfileForm implements java.io.Serializable
{
    private int age;
    private String name;

    public void setName(String name)
    {
        this.name = name;
```

```
    }

    public String getName()
    {
        return this.name;
    }

    public void setAge(int age)
    {
        this.age = age;
    }

    public int getAge()
    {
        return this.age;
    }
}
```

Save and close `ProfileForm.java`.

5.3. To Edit the Controller File to Handle the Submitted Data

Now you will add a new Action and use your new Form Bean to handle the data submitted from the HTML form.

Open the file `C:/beehive_projects/pageflow_tutorial/Controller.java`

Edit `Controller.java` so it appears as follows. Code to add appears in bold type.

`Controller.java`

```
import javax.servlet.http.HttpSession;

...

import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;
import forms.ProfileForm;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    },
    sharedFlowRefs={
        @Jpf.SharedFlowRef(name="shared", type=shared.SharedFlow.class)
    }
)
public class Controller
    extends PageFlowController
{
    @Jpf.Action(
```

```
        forwards = {
            @Jpf.Forward(name="success", path="page2.jsp")
        }
    )
    public Forward processData(ProfileForm form)
    {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());
        return new Forward("success");
    }

    @Jpf.SharedFlowField(name="shared")
    private shared.SharedFlow sharedFlow;

    /**
     * Callback that is invoked when this controller instance is created.
     */
    protected void onCreate()
    {
    }

    /**
     * Callback that is invoked when this controller instance is destroyed.
     */
    protected void onDestroy(HttpSession session)
    {
    }
}
```

Save `Controller.java`.

5.4. To Recompile and Redeploy the Page Flow

Compile and deploy the Page Flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

5.5. To Test the Page Flow Web Application

Visit the following link:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the index.jsp page.

Click the link.

You will be directed to page2.jsp.

Enter values in the Name and Age fields, and click Submit.

Notice the name and age values you entered are displayed in the Tomcat console shell.

6. Step 5: Processing and Displaying Data

6.1. To Create a JSP Page to Display Submitted Data

In this step you will create a new JSP to present the results from processing the data submission.

In the directory `C:/beehive_projects/pageflow_tutorial` create a file named **`displayData.jsp`**.

Edit `displayData.jsp` so it appears as follows.

`displayData.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>displayData.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      You submitted the following information:
    </p>
    <p>
      Name: <netui:content value="${requestScope.data.name}"/>
      <br/>
      Age: <netui:content value="${requestScope.data.age}"/>
    </p>
  </netui:body>
</netui:html>
```

Save and close `displayData.jsp`.

6.2. To Process the Submitted Data

Edit the `processData` method in the `Controller.java` file so it appears as follows. Code to add appears in bold.

`Controller.java`

```
    ...
```

```
    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        }
    )
    public Forward processData(ProfileForm form)
    {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());
        getRequest().setAttribute("data", form);
        return new Forward("success");
    }

    ...
```

Save `Controller.java`.

6.3. To Recompile and Redeploy the Page Flow

Compile and deploy the Page Flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

6.4. To Test the Page Flow Web Application

Visit the following link:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the index.jsp page.

Click the link.

You will be directed to page2.jsp.

Enter values in the Name and Age fields. Click the Submit button.

You will be forwarded to the displayData.jsp page. Notice the values you entered are displayed.

7. Step 6: Input Validation

7.1. To Add Declarative Validation to the Page Flow

In this step you will use declarative validation to define the set of rules for each field, to be applied during input validation. Add a `ValidatableProperty` for the name field of the

form so that it will (1) be a required field and (2) have a maximum length of 30 characters. The age field will also be required and must have a value in the range 0 to 130.

Open the file `C:/beehive_projects/pageflow_tutorial/Controller.java`

Edit the `@Jpf.Action` annotation for the `processData` method in the `Controller.java` file so it appears as follows. Code to add appears in bold. Don't forget the comma after the `forwards={...}` element!

`Controller.java`

```
...

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        },
        validatableProperties = {
            @Jpf.ValidatableProperty(
                propertyName = "name",
                displayName = "Name",
                validateRequired = @Jpf.ValidateRequired(),
                validateMaxLength = @Jpf.ValidateMaxLength(chars = 30)),
            @Jpf.ValidatableProperty(
                propertyName = "age",
                displayName = "Age",
                validateRequired = @Jpf.ValidateRequired(),
                validateRange = @Jpf.ValidateRange(minInt = 0, maxInt =
130))
        },
        validationErrorForward =
            @Jpf.Forward(name="fail",
navigateTo=Jpf.NavigateTo.currentPage)
    )
    public Forward processData(ProfileForm form)
    {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());
        getRequest().setAttribute("data", form);
        return new Forward("success");
    }

    ...
```

Save `Controller.java`.

7.2. To Modify the JSP to Display Validation Errors

Add the `<netui:error>` tag to display validation error messages on the page.

Edit the file `C:/beehive_projects/pageflow_tutorial/page2.jsp` so it appears as follows.

page2.jsp

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
    <p>
      <netui:form action="processData">
        Name: <netui:textBox dataSource="actionForm.name"/>
        <netui:error key="name"/>
        <br/>
        Age: <netui:textBox dataSource="actionForm.age"/>
        <netui:error key="age"/>
        <br/>
        <netui:button type="submit" value="Submit"/>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save and close `page2.jsp`.

7.3. To Recompile and Redeploy the Page Flow

Compile and deploy the Page Flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

7.4. To Test the Page Flow Web Application

Visit the following link:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the index.jsp page.

Click the link.

You will be directed to page2.jsp.

Leave the Name field empty and enter a negative integer value in the Age field. Click the

Submit button.

You will be returned to the page2.jsp page. Notice the error messages for the values you entered.

8. Step 7: Collect Data from a Nested Page Flow

8.1. To Link to the Nested Page Flow

Modify the HTML form tag to add a new data field and a button and link it to an Action in the nested Page Flow. In a later step, the new nested Controller file will be created to handle the data collection.

Edit the file `C:/beehive_projects/pageflow_tutorial/page2.jsp` so it appears as follows. Code to add appears in bold.

`page2.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>page2.jsp</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to page2.jsp!
    </p>
    <p>
      <netui:form action="processData">
        Name: <netui:textBox dataSource="actionForm.name"/>
        <netui:error key="name"/>
        <br/>
        Age: <netui:textBox dataSource="actionForm.age"/>
        <netui:error key="age"/>
        <br/>
        Sport: <netui:textBox dataSource="actionForm.sport"/>
        <br/>
        <netui:button type="submit" action="getSport" value="Select
Sport"/>
        <netui:button type="submit" value="Submit"/>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save `page2.jsp`.

8.2. To Update the Form Bean

In this step you will update the Java class that represents the submission form with the additional data field created in the previous task. When the nested Page Flow returns, the new member of the Form Bean class instance can be loaded with the value collected.

Edit `C:/beehive_projects/pageflow_tutorial/WEB-INF/src/forms/ProfileForm.java` and add the following member variable and methods.

`ProfileForm.java`

```
    ...

    private String sport;

    public void setSport(String sport)
    {
        this.sport = sport;
    }

    public String getSport()
    {
        return this.sport;
    }

    ...
```

Save and close `ProfileForm.java`.

8.3. To Launch and Return from the Nested Page Flow

Add Action methods to handle forwarding to the nested Page Flow and another to implement the return Action when the nested Page Flow completes.

Open the file `C:/beehive_projects/pageflow_tutorial/Controller.java`

Edit `Controller.java` so it appears as follows. Code to add appears in bold type. Don't forget to add the `useFormBean` property to the `@Jpf.Action` annotation of the `processData` method. The ProfileForm is Page Flow scoped for this example, using the same Form Bean instance in multiple Action methods.

`Controller.java`

```
import javax.servlet.http.HttpSession;

import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;
```

```
import forms.ProfileForm;

@Jpf.Controller(
    simpleActions={
        @Jpf.SimpleAction(name="begin", path="index.jsp"),
        @Jpf.SimpleAction(name="toPage2", path="page2.jsp")
    },
    sharedFlowRefs={
        @Jpf.SharedFlowRef(name="shared", type=shared.SharedFlow.class)
    }
)
public class Controller
    extends PageFlowController
{
    private ProfileForm profileForm;

    /**
     * This action forwards to the nested Page Flow to collect the sport
     * name.  Note that it takes a ProfileForm so we can update the form
     * with the sport name returned from the nested Page Flow, but we've
     * explicitly turned validation off for this action, since the form
     * may be incomplete.
     */
    @Jpf.Action(
        useFormBean="profileForm",
        forwards={
            @Jpf.Forward(name="getSportFlow", path="sports/begin.do")
        },
        doValidation=false
    )
    protected Forward getSport(ProfileForm form)
    {
        return new Forward("getSportFlow");
    }

    /**
     * This action takes the sport name returned from the nested Page Flow
     * and updates the field in the form and returns to the original page.
     */
    @Jpf.Action(
        forwards={
            @Jpf.Forward(
                name="success",
                navigateTo=Jpf.NavigateTo.currentPage
            )
        }
    )
    protected Forward sportSelected( String sport )
    {
        profileForm.setSport( sport );
        Forward success = new Forward( "success", profileForm );
        return success;
    }
```

```
    @Jpf.Action(
        useFormBean="profileForm",
        forwards = {
            @Jpf.Forward(name="success", path="displayData.jsp")
        },
        validatableProperties = {
            @Jpf.ValidatableProperty(
                propertyName = "name",
                displayName = "Name",
                validateRequired = @Jpf.ValidateRequired(),
                validateMaxLength = @Jpf.ValidateMaxLength(chars = 30)),
            @Jpf.ValidatableProperty(
                propertyName = "age",
                displayName = "Age",
                validateRequired = @Jpf.ValidateRequired(),
                validateRange = @Jpf.ValidateRange(minInt = 0, maxInt =
130))
        },
        validationErrorForward =
            @Jpf.Forward(name="fail",
navigateTo=Jpf.NavigateTo.currentPage)
    )
    public Forward processData(ProfileForm form)
    {
        System.out.println("Name: " + form.getName());
        System.out.println("Age: " + form.getAge());
        getRequest().setAttribute("data", form);
        return new Forward("success");
    }

    @Jpf.SharedFlowField(name="shared")
    private shared.SharedFlow sharedFlow;

    /**
     * Callback that is invoked when this controller instance is created.
     */
    protected void onCreate()
    {
    }

    /**
     * Callback that is invoked when this controller instance is destroyed.
     */
    protected void onDestroy(HttpSession session)
    {
    }
}
```

Save `Controller.java`.

8.4. To Create a Nested Page Flow

In this step you will create a nested Page Flow with actions to select and confirm the data to

return to the main or nesting Page Flow. The new nested Controller file contains an inner Form Bean classs for the data collection. It has only a single field for the user's choice of sport activity. The options to be displayed are declared as member data of this nested Page Flow. After the user confirms the data, the nested Page Flow returns a `String` to the main Page Flow.

In the directory `C:/beehive_projects/pageflow_tutorial/` create a directory named **`sports`**.

In the directory `C:/beehive_projects/pageflow_tutorial/sports` create a JAVA file named **`SportsController.java`**.

Edit `C:/beehive_projects/pageflow_tutorial/sports/SportsController.java` so it appears as follows.

`SportsController.java`

```
package sports;

import org.apache.beehive.netui.pageflow.FormData;
import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.PageFlowController;
import org.apache.beehive.netui.pageflow.annotations.Jpf;

@Jpf.Controller(
    nested = true,
    simpleActions = {
        @Jpf.SimpleAction(name="begin", path="index.jsp")
    }
)
public class SportsController
    extends PageFlowController
{
    private String selectedSport;
    private String[] sports = {"sailing", "surfing", "diving",
"volleyball",
                               "bicycling"};

    public String[] getSports() {
        return sports;
    }

    public String getSelectedSport() {
        return selectedSport;
    }

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(name="confirm", path="confirm.jsp")
        }
```

```
    )
    public Forward selectSport(SportForm form)
    {
        selectedSport = form.getSport();
        return new Forward("confirm");
    }

    @Jpf.Action(
        forwards = {
            @Jpf.Forward(
                name="success",
                returnAction="sportSelected",
                outputFormBeanType=String.class)
        }
    )
    public Forward confirm()
    {
        return new Forward("success", selectedSport);
    }

    public static class SportForm extends FormData
    {
        private String sport;

        public void setSport(String sport)
        {
            this.sport = sport;
        }

        public String getSport()
        {
            return this.sport;
        }
    }
}
```

Save and close `SportsController.java`.

8.5. To Present and Collect Data using a Form

This step illustrates the use of custom tags to render a radio button group in an HTML form and link it to the nested Page Flow `selectSport` Action method.

In the directory `C:/beehive_projects/pageflow_tutorial/sports`, create a file named `index.jsp`.

Edit index.jsp so it appears as follows.

`index.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
```

```
<netui:html>
  <head>
    <title>Select Sport</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Select Sport Activity
    </p>
    <p>
      <netui:form action="selectSport">
        <table>
          <tr>
            <td>Sports:</td>
            <td>
              <netui:radioButtonGroup dataSource="actionForm.sport"
                      optionsDataSource="${pageFlow.sports}"/>
            </td>
          </tr>
        <table>
        <netui:button type="submit">Submit</netui:button>
      </netui:form>
    </p>
  </netui:body>
</netui:html>
```

Save `index.jsp`.

8.6. To Confirm the Selected Data

In the directory `C:/beehive_projects/pageflow_tutorial/sports`, create a file named `confirm.jsp`.

Edit confirm.jsp so it appears as follows.

`confirm.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Confirm Sport Activity</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Confirm Sport Activity
    </p>
    <p>
      Sport: <netui:content value="${pageFlow.selectedSport}"/>
    </p>
    <netui:form action="confirm">
```

```
        <netui:button type="submit" value="Confirm"/>
    </netui:form>
  </netui:body>
</netui:html>
```

Save `confirm.jsp`.

8.7. To Recompile and Redeploy the Page Flow

Compile and deploy the Page Flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

8.8. To Test the Page Flow Web Application

Visit the following link:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the index.jsp page.

Click the link.

You will be directed to page2.jsp.

Click the "Select Sport" button.

You will be directed to sports/index.jsp in the nested Page Flow.

Click a radio button and then the Submit button.

You will be directed to sports/confirm.jsp in the nested Page Flow.

Click the Confirm button.

You will be returned to the page2.jsp page. Notice that the value you selected is displayed in the Sport field.

9. Step 8: Adding Actions to a Shared Flow

9.1. To Create a Common Destination JSP Page

In the directory `C:/beehive_projects/pageflow_tutorial`, create a file named `help.jsp`.

Edit help.jsp so it appears as follows.

`help.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<netui:html>
  <head>
    <title>Help Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      Welcome to the Help Page
    </p>
  </netui:body>
</netui:html>
```

Save `help.jsp`.

9.2. To Make an Action available to multiple Page Flows

In this step you will add a Simple Action to the existing Shared Flow. The Action forwards to the help page created in the previous step and will be available to multiple Page Flows in the application.

Open the existing Shared Flow file
`C:/beehive_projects/pageflow_tutorial/WEB-INF/src/shared/SharedFlow.java`

Edit the `@Jpf.Controller` annotation for the Shared Flow controller class, `SharedFlow`, in the `SharedFlow.java` file and add the `simpleActions` property. Code to add appears in bold. Don't forget the comma after the `catches={...}` element!

`SharedFlow.java`

```
...

@Jpf.Controller(
    catches={
       @Jpf.Catch(type=PageFlowException.class,
method="handlePageFlowException"),
       @Jpf.Catch(type=Exception.class, method="handleException")
    },
    simpleActions={
        @Jpf.SimpleAction(name="showHelp", path="/help.jsp")
    }
)
public class SharedFlow
    extends SharedFlowController
{
```

```
    ...
```

Save `SharedFlow.java`.

9.3. To Link a Page to the Shared Flow Action

In this step you will create a link from the JSP, `index.jsp` to the Shared Flow Action.

Open the file `C:/beehive_projects/pageflow_tutorial/index.jsp`.

Edit `index.jsp` so it appears as follows. The code to add appears in bold type.

`index.jsp`

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0"
prefix="netui-data"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-html-1.0"
prefix="netui"%>
<%@ taglib uri="http://beehive.apache.org/netui/tags-template-1.0"
prefix="netui-template"%>
<netui:html>
  <head>
    <title>Web Application Page</title>
    <netui:base/>
  </head>
  <netui:body>
    <p>
      New Web Application Page
    </p>
    <p>
      <netui:anchor action="toPage2">Link to page2.jsp</netui:anchor>
    </p>
    <netui:anchor action="shared.showHelp" popup="true">Help
      <netui:configurePopup location="false" width="550" height="150">
      </netui:configurePopup>
    </netui:anchor>
  </netui:body>
</netui:html>
```

Save `index.jsp`.

9.4. To Recompile and Redeploy the Page Flow

Compile and deploy the Page Flow using the same Ant and copy commands used in step 2.

If you are asked to overwrite the old WAR file, enter 'Yes'.

Wait a few seconds for Tomcat to redeploy the WAR file, then move on to the next step.

9.5. To Test the Page Flow Web Application

Visit the following link:

http://localhost:8080/pageflow_tutorial/begin.do

You will be directed to the index.jsp page.

Click the help link.

A popup window with the help.jsp page will be displayed.